Autonimic energy-aware task scheduling

Tom Guérout, Mahdi Ben Alaya

Autonomic energy-aware tasks scheduling

Tom Guérout
CNRS, LAAS, 7 avenue du Colonel Roche,
F-31400 Toulouse, France
IRIT/Toulouse University, 118 Route de Narbonne,
F-31062 TOULOUSE CEDEX 9
Univ de Toulouse, INSA, LAAS,
F-31400 Toulouse, France
Email: tguerout@laas.fr

Mahdi Ben Alaya
CNRS, LAAS, 7 avenue du Colonel Roche,
F-31400 Toulouse, France
Univ de Toulouse, INSA, LAAS,
F-31400 Toulouse, France
Email: ben.alaya@laas.fr

***Abstract*—The increasing processing capability of data-centers increases considerably their energy consumption which leads to important losses for companies. Energy-aware task scheduling is a new challenge to optimize the use of the computation power provided by multiple resources. In the context of Cloud resources usage depends on users requests which are generally unpredictable. Autonomic computing paradigm provides systems with self-managing capabilities helping to react to unstable situation. This article proposes an autonomic approach to provide energy-aware scheduling tasks. The generic autonomic computing framework *FrameSelf* coupled with the *CloudSim* energy-aware simulator is presented. The proposed solution enables to detect critical schedule situations and simulate new placements for tasks on DVFS enabled hosts in order to improve the global energy efficiency.**

***Index Terms*—Autonomic, Framework, self-management, Simulations, Energy, Efficiency, DVFS**

I. Introduction

The article is based on the use of an autonomic framework and a simulator which allows to compute energy-aware simulations. The aim is to do energy-aware scheduling of tasks. The combination of using these two tools allow to have the benefit of the autonomic middle-ware to detect specific events, and the simulator to obtain informations about energy consumption which help to take a scheduling decision.

Autonomic computing paradigm is a concept based on the human body's autonomic nervous system. A such system manages the functioning of computer applications and systems with minimum human intervention, in the same way that the autonomic nervous system regulates body systems without conscious human input. The goal of autonomic computing is to create autonomous systems, capable of high-level functioning by implementing self-management properties such as self-configuration, self-optimisation, self-healing and self-protection.

The increasing use of data-centers makes the analysis of energy consumption increasingly important. Some metrics to evaluate their efficiency are known (PUE, ERE) [1] and much researches are ongoing to find new ways to reduce energy. Real platforms can be used to lead these phases of validation, but it involves a significant preparation time and physical measurements are not always possible or easy to do depending on the available equipments. That is why simulators are more commonly used in this area. *CloudSim* already has some energy-aware functionalities included in its architecture, and has been chosen to be improved and used to conduct energy-aware DVFS simulations in this article.

This article is organized as follows. First, section II presents a state of the art on energy-aware tools and autonomic frameworks. Section III describes the *CloudSim* simulator, the new implementation of DVFS added and how it is able to do energy-aware simulations. Then, Section IV presents the autonomic framework called *FrameSelf*. In section V, first results obtained using both *FrameSelf* and *CloudSim* are given. Finally, section VI concludes and presents ideas for future work.

II. State of the Art

A. Autonomic tools

Autonomic Computing is a paradigm proposed by IBM in 2001 [2]. It aims at developing distributed system capable of self-management to hide intrinsic complexity to administrators and users. An autonomic manager is organized into four main modules, which are Monitor, Analyzer, Planer and Executor. These modules share the same knowledge (managed resources details, policies, symptoms, request for change, plans, etc.), exploit policies based on goal and environment awareness and constitute together the MAPE-K control loop.

Recent works on autonomic computing addressed some of the self-management capabilities applied in specific domains such as DeployWare [3] for dynamic software deployment, OceanStore [4] for resource allocation reconfiguration, Gryphon [5] for communication patterns adaptation, and Astrolabe [6] for autonomic query processing. Cited autonomic solutions are designed to handle problems in specific domains, and are in most cases highly dependent on the type of managed resources. These framework are not modular and do not support multi-model representations in their knowledge bases for advanced management.

B. Energy-aware tools

Energy-aware tools are solutions, that can be used at different levels, which allow to minimize the power consumption

of hosts in a data-centers. In this section, only the host level is addressed and the three common energy-aware tools are presented, considering that all used hosts enable virtualization.

The first solution, called "ON/OFF" method turns off hosts not enough used (compared to a CPU load threshold), and switched on them again if necessary. In this case any all processes running in this host have to be moved to other host(s), and thus, the host previously underutilized can be turned off. Conversely, when all hosts are over-used and the demand is too high, one or more host are started.

As described just above, it is sometimes necessary to move processes from host to host. This mechanism is called migration [7]. It allows to move a virtual machine (and all its environment) from a host to another. These migrations are not free in terms of energy because every movement requires time and it is also important to take into account the total cost of such action. This technique frees the hosts and then turn them off, in order to try to use operating hosts at their maximum potential (consolidation). Two kinds of consolidations works can be defined on virtual machines placement and virtual machines migration, as it detailed in [8].

Finally, the DVFS (Dynamic Voltage and Frequency Scaling) [9] allows to dynamically change the voltage and the frequency of a host in relation to their CPU load. In the Linux kernel, the DVFS can be activated in five different modes: *Performance*, *PowerSave*, *UserSpace*, *Conservative* and *OnDemand*. Each mode has a governor to decides whether the frequency must be changed (increased or decreased) or not.

Three of these five modes use fixed frequency : *Performance* uses the CPU at its highest frequency, *PowerSave* uses the CPU at its lowest frequency and the *UserSpace* mode allows the user to choose one of all available frequencies.

The two last DVFS modes, *Conservative* and *OnDemand* have a dynamical behaviour. It means that the CPU frequency can vary over time regarding the CPU load.

The governors of these two modes work with thresholds (one or two) and periodically check whether the CPU load is lower (or higher) than these thresholds before taking their decision to change the current frequency. The *Conservative* governor works with an *up_threshold* and a *down_threshold*, when the CPU load is higher than the *up_threshold* the frequency is increased, when the CPU load is under the *down_threshold* the CPU frequency is decrease. This mode is very progressive, and each CPU frequency is done step by step through all available frequencies. The *OnDemand* mode uses only one threshold and favours the performance by directly set the fastest CPU frequency when the CPU load exceeds the threshold. A decreasing CPU frequency happens if the CPU load stays below the threshold for a while.

A lower frequency reduced the CPU power consumption, it slows down an application consuming a lot of calculation, but without affect the time spent in I/O or communication.

III. Energy-aware simulations

This section presents the features developed and incorporated in the *CloudSim* simulator in order to support energy-aware simulations using DVFS. CloudSim has been chosen for this work because it is composed of many energy-aware tools needed to run reliable simulations. Some other powerful simulators like SimGrid [10], GSSIM [11] [12] or GreenCloud [13] have been analysed and taken into account, but they do not collect enough energy-aware tools needed for this work.

A. CloudSim *overview*

CloudSim [14] is a Cloud simulator based on GridSim [15]. CloudSim is a toolkit for modeling and simulation of Infrastructure as a Service (IaaS) cloud computing environments. The data-centers architecture is well modeled and can be easily modified by the user according to its needs. It allows users to define the characteristics of data-centers, including number and characteristics of hosts, available storage, network topology, and patterns of data-centers usage. *CloudSim* allows the virtualization, development of scheduling policies, network simulation between data-centers as well as compute the cost of a IaaS cloud computing service. Energy functionalities of *CloudSim* allow to turn off hosts, migrate virtual machines and the integration of energy models.

In order to have all energy tools needed in this simulator, the DVFS has been implemented and added in the core of the simulator.

B. *DVFS integration*

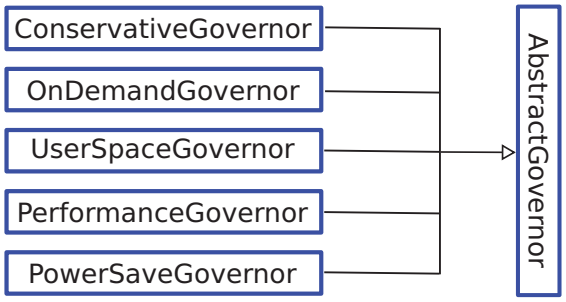

Fig. 1. Classes of the new DVFS package added in the CloudSim simulator

Governors of the five DVFS modes, as they are present in the Linux kernel have been implemented. Their role is to determine whether the CPU frequency must be modified, and their decision is directly related to their intrinsic decision rule. A frequency change directly impacts the CPUs capacity in MIPS (*Millions Instructions Per Second*) and inexorably the total capacity of the host. Indeed, the new DVFS package (Figure 1) is composed of one *abstract* class which contains all common parameters of the different modes, and each specific mode's behaviour is described in the five other distinct classes. This package is closely linked with CPU and Host classes of CloudSim, so as to allow the analysis and the control of their load, frequencies and capacity states.

It also involves how the simulator has to manage the virtual machines capacities. For example, if the system decides to reduce the frequency, the sum of the capacities of all virtual machines can temporarily exceed the maximum capacity of

the host. In this case, the size of each virtual machine must be decrease in proportion to the new host capacity to fit them in this host. The same situation occurs when one or more new virtual machines are created and need to be hosted. If the sum of capacities of all virtual machines running and those of the new virtual machines created exceeds the maximum capacity of the host the same process of reducing the capacity of virtual machines is applied.

Also, when part of the capacity of a host is freed, the capacity of virtual machines still running can be increased. This event occurs when a virtual machine finishes its execution or when the CPU frequency is increased, all virtual machines sizes are increased in proportion to the free capacity of the host, while taking care to not exceed their maximum available capacity.

These different situations that imply to dynamically modify virtual machines capacities regarding the frequency changes also mean that the DVFS package has been implemented so as to be closely linked to the virtualization layer and the main core classes of CloudSim.

Finally, an input XML configuration DVFS file has been added to allow the user to specify:

- the available frequencies of the CPU
- which DVFS mode has to be activated
- thresholds' values to be used (for dynamic modes)

C. Simulations using dynamic DVFS policy

For these simulations the dynamic *OnDemand* mode is used. Its intrinsic behaviour allows to reduce the CPU frequency during an *Idle* CPU phase, and set it to its maximum when a CPU burn phase comes. These frequency changes are managed by the *OnDemand* governor and give low power consumption when the host is under-used. This governor also favours the performance of the host when the full CPU computation power is needed. Indeed, it sets the CPU frequency at the maximum available value as soon as the CPU load overstep a threshold (95% of CPU load). Conversely, when the CPU load stay under this threshold, the CPU frequency is decreased step by step using all frequencies available in the host. If a host is temporally under-used, but not for long enough to decide to move its tasks, this dynamic mode allows to reduce the energy consumption while keeping this host switched on.

D. How are simulations used

In the autonomic context, two types of events can be defined. The first one concerns the arrival of a new task that need to be schedule in one host. The other event happens when a host is under-used regarding a CPU load threshold (15% for example). These two cases lead to trigger an autonomic event, in the first case the aim is to find the best host in term of energy consumption to execute the task, in the second one it is necessary to freeing the host (stop and move all processes) by migrating all tasks to other hosts. Of course, these moves are not free in term of time, due to the time need to stop processes, copy data from one host to another(s) and then continue the execution on new chosen hosts. In term of energy consumption this lead to switch off one host, indeed saving energy, and consolidate the others that already have enough running processes to keep them switched on.

In these two described cases, simulations are dynamically used in order to find the best energy efficiency scheduling of tasks.

The next section introduces the autonomic framework, Frameself.

IV. Frameself autonomic framework

The Frameself [16] framework is a generic autonomic manager based on the IBM autonomic architecture reference [17]. The monitor, analyzer, planer, and executer operate as expert systems to emulate the decision-making ability of human experts. These modules are independent from managed resources and are designed to solve complex problems by reasoning about knowledge, like an expert. Each module is divided into two parts, one fixed, independent of the system: the inference engine, and one variable: the knowledge base model. The Frameself global architecture is described in Figure 2.

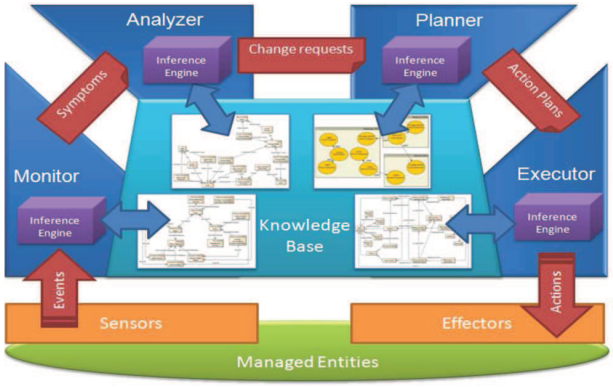

Fig. 2. Frameself architecture overview

A. Main components overview

The monitor collects events from different sensors and transforms, extends and formats them in a standardized format to make them consistent for processing. It discards events that are deemed to be irrelevant for the management platform and ignores events pertaining to systems that are downstream of a failed resource. Masking is different from filtering because it allows to dynamically hide unfiltered events according to the environment changes. The monitor also merges duplicates of the same event that may be caused by network instability. It infers symptoms based on received events using knowledge models then publish generated symptoms on the message bus to make them available to the analyzer. The monitor acts as a proxy to connect specific event collectors that depends on existing sensors. Depending on the targeted system, a specific event collector can be an asynchronous event subscriber, a synchronous event requester, or an even log file parser.

The analyzer subscribes to the message bus and receives published symptoms. It infers Requests For Change (RFCs) based on received symptoms and knowledge models, then publishes them on the message bus in order to make them available to the planner. The analyzer can also check and validate a policy entered by the administrator and can resolve conflicts between policies.

The planner subscribes to the message bus and receives published Plans. It reads information related to policies and available effectors from the knowledge base, generates action plans leading to received RFCs, then publishes generated plans on the message bus to make them available to the Executer. The planner can also convert high-level policies into low-level ones and employ them to guide the decisions.

The executer subscribes to the message bus and receives published plans. It interprets received actions and checks whether it has the rights and means to execute them. It schedules and orchestrates the execution process by describing the automated arrangement and coordination of actions in time, then executes them on distributed destinations. The executer acts as a proxy to connect specific action dispatchers that depend on existing effectors. Based on the targeted system, a specific action dispatcher can be a web-service client able to request web-services or a deployment manager able to send and execute script files into distributed machine.

The Knowledge Base allows to maintain all essential informations to execute self-management operations on the managed system. Knowledges can be divided in different categories :

- Topology of the system architecture.
- Descriptions of sensors and effectors.
- Data about symptoms, RFCs, actions and used policies.

Other domain specific models such as ontology or graphs, and reasoning rules can be instantiated, within the knowledge base, independently for each control loop component. Thus, each model can be handled in a decoupled way which enables to manage a multi-model knowledge base.

B. Functioning description

The monitor subscribes to available sensors and receives events such as a managed resource "CPU_usage", or "New_task_arrival". Collected events are normalized, filtered, aggregated, and then inserted in the knowledge base monitoring session.

The monitor applies business rules using the Drools inference engine [18] to generate relative symptoms such as "Low_CPU" or "NonScheduled_task" and sends them to the analyzer. This latter provides the mechanisms that correlate and model complex situations. These mechanisms allow the autonomic manager to learn about the environment and help to predict environment changes. It receives symptoms as input, generates new knowledge about required requests for change (RFCs) such as "deploy_task" or "migrate_task" and sends them for planning.

The planner acts as a decision module for selecting the appropriate hosts to execute tasks. It saves received RFCs as goal states, reads models of possible actions and facts from the knowledge base and applies an energy efficiency policy to guide its work. The planner generates actions such as "1_install_task" then "2_starting_task" for task deploying, or "1_stop_task", "2_delete_task", "3_install_task", then "4_start_task" for task migration. The executor performs the received plan using effectors and controls the actions execution with consideration for dynamic updates.

The next section presents first results obtained both on autonomic management and energy-aware scheduling.

V. First results

Performance results are experimented to calculate the overload that *FrameSelf* generates to handle an increasing number of events. Obtained time values include monitoring and analysing processes based on the Drools inference engine, as well as the planning process based on the *CloudSim* energy-aware simulator. Figure 3 shows the *FrameSelf* self-management capacity calculated after receiving and handling new task arrival events from a number of 10 to 1000 events.

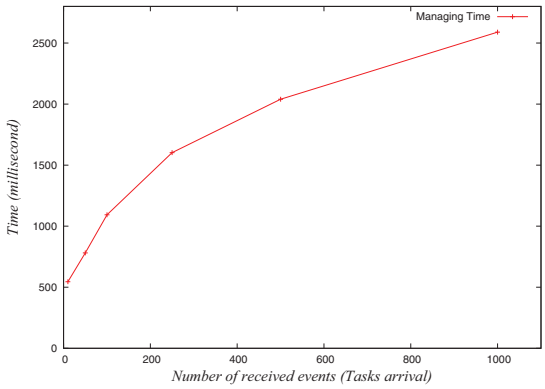

Fig. 3. *FrameSelf* self-managing capacity using 20 hosts.

Scheduling results obtained with *CloudSim* have been done using a metric that allow to classify hosts by their power consumption characteristic related to their free capacity.

The metric considered to be minimized, called *Watt_Per_Mips* (WPM), is defined as follow:

Lets denote the considered set of hosts by $H_j = \{H_1, ..., H_n\}$ with a maximum of Nb_H elements, then for each host each set of CPUs can be defined as $H_j^{cpus} = \{cpu_1^{H_j}, ..., cpu_n^{H_j}\}$ with a maximum of Nb_{CPU} elements. Finally, the set of virtual machines running on these hosts are defined by $VM_k = \{VM_1, ..., VM_n\}$ with a maximum of Nb_{VM} elements.

Then, power and capacity of hosts and CPUs are defined as:

$P_{min}^{H_j}$ and $P_{max}^{H_j}$: the powers deliver by an host H_j at *Idle* (0%) and *Full* (100%) states.

$P_{min}^{cpu_i}$ and $P_{max}^{cpu_i}$: the powers deliver by a CPU cpu_i of a host H_j ($cpu_i \in H_j^{cpus}$) at *Idle* (0%) and *Full* (100%) states.

Each capacity of Host, CPU and virtual machines are defined as:

$M_{curr}^{H_j}$: the current capacity of the host H_j (in MIPS).

$M_{max}^{H_j}$: the maximum capacity of the host H_j (in MIPS).

$M_{curr}^{cpu_i}$: the current capacity of the CPU cpu_i (in MIPS).

$M_{max}^{VM_k}$: the maximum capacity of virtual machine VM_k (in MIPS).

Lets define minimum and maximum powers (at *Idle* and *Full* states) given by the CPUs as:

$$P_{min}^{cpu_i} = \frac{P_{min}^{host_j} \times M_{curr}^{cpu_i}}{M_{curr}^{H_j}} \text{ and } P_{max}^{cpu_i} = \frac{P_{max}^{host_j} \times M_{curr}^{cpu_i}}{M_{curr}^{H_j}}$$

Finally, the WPM metric computed for each host H_j is equal to:

$$\text{WPM} = \frac{\sum_{i=1}^{nbCPU} \left((P_{max}^{cpu_i} - P_{min}^{cpu_i}) \times \frac{\sum_{k=1}^{Nb_{VM}} M_{max}^{VM_k}}{M_{max}^{H_j}} \right)}{Nb_{CPU}} \quad (1)$$

This metric allows to find the best *green* host at a given time t, by taking into account each current CPU frequency (defined in MIPS in equations), the potential of a host to accommodate new tasks (free capacity), and the powers values of the host using these current CPUs frequencies.

The scheduling time depends on how many hosts are taken into account and how many tasks have to be scheduled. Results using this metric (eq 1) have been done with 20 hosts and is depicted on Figure 4. Results on this figure have been obtained with CloudSim by simulating a tasks scheduling, from a number of 10 to 1000 tasks, with the same arrival time.

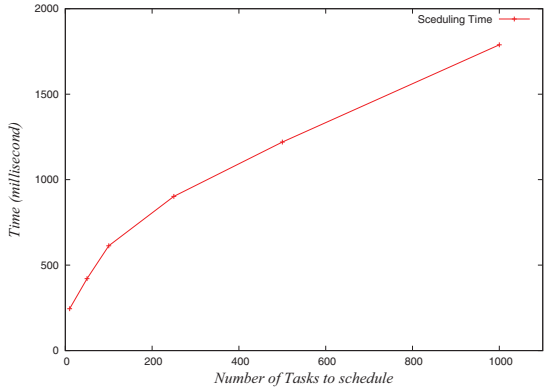

Fig. 4. *CloudSim* scheduling time using 20 hosts.

VI. Conclusion

This article presents first ideas and first results of using both autonomic framework and an energy-aware simulator. The aim is to use the decision rules bases of the autonomic manager to schedule and migrate tasks. Two critical situations are detected such a new task arrival, so needs to be schedule and an under-used host needed to be freeing. The autonomic framework is used for energy efficiency self-optimization and the simulator is used to be able to have estimations of the power consumption using different scheduling policies in order to take an energy efficient scheduling decision.

Proposed perspectives include new experiments using more complex scenario, by multiplying the number of type of events and using a real platform architecture model to involve a higher number of hosts in order to validate the scalability of this approach. Another perspective is to adapt this approach to a real scheduler, as OAR [19], in Grid and Cloud infrastructures.

References

[1] M. Patterson, B. Tschudi, O. Vangeet, J. Cooley, and D. Azevedo, "ERE: A metric for measuring the benefit of reuse energy from a data center," The Green Grid, Tech. Rep. White Paper 29, 2010.

[2] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[3] A. Flissi, J. Dubus, N. Dolet, and P. Merle, "Deploying on the grid with deployware," in *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*. IEEE, 2008, pp. 177–184.

[4] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer *et al.*, "Oceanstore: An architecture for global-scale persistent storage," *ACM Sigplan Notices*, vol. 35, no. 11, pp. 190–201, 2000.

[5] P. R. Pietzuch and S. Bhola, "Congestion control in a reliable scalable message-oriented middleware," in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. Springer-Verlag New York, Inc., 2003, pp. 202–221.

[6] K. P. Birman, R. Van Renesse, and W. Vogels, "Scalable data fusion using astrolabe," in *Information Fusion, 2002. Proceedings of the Fifth International Conference on*, vol. 2. IEEE, 2002, pp. 1434–1441.

[7] C. C. Keir, C. Clark, K. Fraser, S. H, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005, pp. 273–286.

[8] E. Feller, "Autonomic and energy-efficient management of large-scale virtualized data centers," Ph.D. dissertation, Université Rennes 1, 2012.

[9] T. Kolpe, A. Zhai, and S. Sapatnekar, "Enabling improved power management in multicore processors through clustered dvfs," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, march 2011, pp. 1 –6.

[10] H. Casanova, A. Legrand, and M. Quinson, "Simgrid: A generic framework for large-scale distributed experiments," in *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, april 2008, pp. 126–131.

[11] S. Bak, M. Krystek, K. Kurowski, A. Oleksiak, W. Piatek, and J. Waglarz, "Gssim-a tool for distributed computing experiments." *Scientific Programming*, vol. 19, no. 4, pp. 231–251, 2011.

[12] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz, "Grid scheduling simulations with gssim," in *Parallel and Distributed Systems, 2007 International Conference on*, vol. 2, dec. 2007, pp. 1–8.

[13] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. Khan, "Greencloud: A packet-level simulator of energy-aware cloud computing data centers," in *GLOBECOM 2010, IEEE Global Telecommunications Conference*, 2010, pp. 1–5.

[14] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[15] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *concurrency and computation: practice and experiene (ccpe)*, vol. 14, no. 13, pp. 1175–1220, 2002.

[16] M. Ben Alaya and T. Monteil, "Frameself: A generic context-aware autonomic framework for self-management of distributed systems," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*. IEEE, 2012, pp. 60–65.

[17] M. Parashar and S. Hariri, *Autonomic computing: concepts, infrastructure, and applications*. CRC, 2006.

[18] M. Proctor, "Relational declarative programming with jboss drools," in *Symbolic and Numeric Algorithms for Scientific Computing, 2007. SYNASC. International Symposium on*. IEEE, 2007, pp. 5–5.

[19] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard, "A batch scheduler with high level components," in *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, vol. 2, may 2005, pp. 776 – 783 Vol. 2.